(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
21 December 2000 (21.12.2000)

PCT

(10) International Publication Number
WO 00/77987 A1

(51) International Patent Classification⁷: H04L 12/58, G06F 17/60

(21) International Application Number: PCT/IL99/00318

(22) International Filing Date: 14 June 1999 (14.06.1999)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant *(for all designated States except US)*: **ACTIVE-NAMES LTD.** [IL/IL]; 30 Kalisher Street, 65257 Tel-Aviv (IL).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **MIMRAN, David** [IL/IL]; 4/6 Nes Ziona Street, 58336 Holon (IL). **STEINBOCK, Nimrod** [IL/IL]; 31/a Balfour Street, 65211 Tel-Aviv (IL). **RAZ, Arbel** [IL/IL]; 147/6 Jabotinsky Street, 62150 Tel-Aviv (IL).

(74) Agent: **REINHOLD COHN AND PARTNERS**; P.O.Box 4060, 61040 Tel-Aviv (IL).

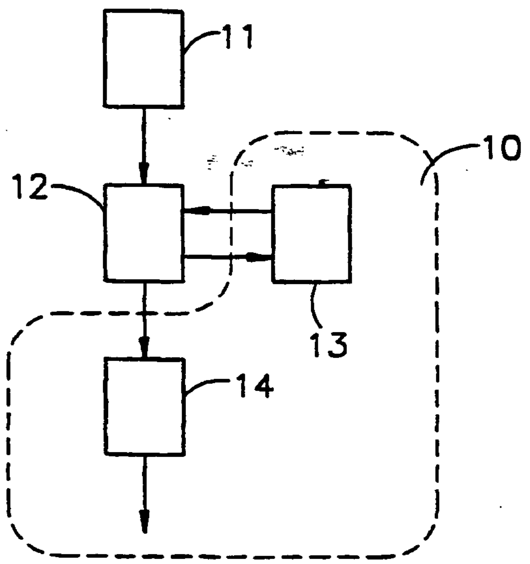
(81) Designated States *(national)*: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *With international search report.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: METHOD FOR RESOLVING ELECTRONIC ADDRESSES IN DATA-COMMUNICATIONS



(57) Abstract: A method for resolving electronic addresses in data-communications wherein users register their address-related details with a virtual centralized location in a topology of servers (10), such as a centralized server (13) or any communication topology equivalent server architecture. These users also acquire (for example, by an Internet download) special software (e.g. an "agent") (12) that is installed on their computer. The "agent" (12) intercepts outgoing electronic data-communications messages at the user's computer, consult the virtual centralized location in a topology of servers for the validity of the recipient address, and replace the address with the valid address, if such replacement is needed.

WO 00/77987 A1

Method f r Resolving Electronic Addresses in Data-Communications

FIELD OF THE INVENTION

The present invention relates to Data-Communications. More specifically, the present invention relates to electronic mail, Internet, Internet Addresses, and Simple Mail Transfer Protocol (SMTP) used in wide area and like networks.

BACKGROUND OF THE INVENTION

Presently, there are problems related to changes to electronic data-communications (e.g. e-mail) addresses, which often result in lost electronic data-communications messages.

An existing treatment to the problem is given by "mail forwarding" which has major drawbacks. Mail forwarding exposes mail content to third party inspection. Mail forwarding forces a user to get a new electronic data-communications address, which in turn causes a burden of use to be inflicted on new users to the system, especially on their contacts. Worst of all, this burden occurs every time the user changes address.

There is a need in the art for a method of substantially accomplishing the functions of the "mail forwarding" albeit without the drawbacks, especially third party exposure of data-communications content.

SUMMARY OF THE INVENTION

The present invention generally relates to a method for resolving electronic addresses in data-communications. Users of the present invention will have to register their address-related details with a virtual centralized location in a topology of servers, such as a centralized server or any communication topology equivalent server architecture. Users will have to acquire (for example, by an Internet download) special software (e.g. an "agent") that will be installed on their computer. The "agent" will intercept outgoing electronic data-communications messages at the user's computer, consult the virtual centralized location in a topology of servers for the validity of the recipient address, and replace the address with the valid address, if such replacement is needed.

More specifically the present invention relates to embodiments accomplishing and actualizing a method for directing electronic data-communications to a current target address, the method including the steps of:

a) at a virtual centralized location in a topology of servers, **establishing** a list of registered addresses;

b) each computer or a proxy processor thereto, of at least one user computers, **acquiring** a client software agent;

c) for substantially each occurrence of a user computer of the at least one user computers **sending** an electronic data-communications, the acquired client, software agent **executing** an address substitution transaction including a verification query to the list of registered addresses; and

d) for substantially each occurrence of the virtual centralized location in a topology of servers receiving a verification query from a user computer of the at least one user computers, the virtual centralized location in a topology of servers **executing** a response transaction, by:

i) **extracting** each address of the at least one intercepted addresses in the verification query,

ii) for substantially each extracted address, **finding** a current target address in the list of registered addresses,

iii) **forming** a query response of address acknowledgements substantially from the found current target addresses, and

iv) **transmitting** the formed query response to the user computer who originated the present verification query.

In the context of the present invention, "electronic data-communications addresses" relates to private and public addresses in a Wide Area Network, a Local Area Network, or the like, wherein these addresses are used for routing electronic mail (email), downloading files, viewing home pages, or the like. According to an embodiment of the present invention, a current target address in the list of registered addresses is a mailing list having at least one address of the type that are used for routing electronic mail, downloading files, viewing home pages, or the like.

In the context of the present invention, a virtual centralized location in a topology of servers is any location in a communications system for interconnecting computers wherein the appearance of a single centralized "authority" is presented. This may actually be a centralized server. Alternatively it may be a distributed process or set of data or protocol on any communication topology-equivalent server architecture whereby a functional equivalence is achieved with respect to an interaction with a user computer.

Since intrinsically each computer or proxy thereto is interconnected to the virtual centralized location in a topology of servers, the actual location of an agent for the user computer may be "physically located" anywhere on a data-communications linkage between the two (e.g. on a file server, on a mail server SMTP (as a plug in) etc. The user-associated agent may actually be a part of the user computer. The agent may be located anywhere on a path between a user computer (sender) and the target of the sending (receiver).

Alternatively, the user-associated agent may actually be a part of the virtual centralized location in a topology of servers, such as on a server associated with that user computer. Likewise, the agent may imitate the user computer from the perspective of the server, and may imitate the server from the perspective of the user computer.

According to the preferred embodiment of the present invention, for segments not containing an electronic data-communications address, the client software agent passes segments to a server using a simple mail protocol. Simply stated, this means that in a segmented (long) electronic data-communications only the segments containing at least one address may be subjected to an address substitution or verification transactions. All other segments are transparently passed through the agent.

According to the preferred embodiment of the present invention, a user's view of acquired client software agent operation emulates SMTP server operation. Simply stated, this means that most data-communications are processed by the agent in a manner transparent to the user.

According to an embodiment of the present invention, a user's view of acquired client software agent operation emulates a plug-in electronic data-communications software package's operation.

Notwithstanding the method as heretofore described, an agent is never required to use a response from the virtual centralized location in a topology of servers, nor is an agent required to wait for an untimely response from the virtual centralized location in a topology of servers. An embodiment of the present invention giving control over the agents actions to the user computer is preferred in the public domain (e.g. internet), while an embodiment of the present invention giving control to a centralized topological location is preferred in the private domain (e.g. intranet).

According to an embodiment of the present invention, (step a) **establishing** a list of registered addresses includes **managing** at least one address therein according to time-of-day, day-of-week, date-in-year, or a personal schedule. This feature allows an owner of a target address to specify the address that he wants his email sent to according to his availability. For example a target address substitution may be to one email address during working hours and to another email address during non-working hours (evenings, weekends, and holidays).

According to the preferred embodiment of the present invention, (step b) **acquiring** a client software agent includes **transacting** a **registering** of a current target address and of other predetermined data for a user at a user computer. Simply stated, this means that at the same time that the "agent" is downloaded there is a registering of at least one user who wants to allow others to use the address forwarding service for his own address.

Furthermore, according to the preferred embodiment of the present invention, the client software agent **transacting** includes, for a user at a user computer, the virtual centralized location in a topology of servers **updating** of a registered current target address of other predetermined data in the established list of registered addresses. While the most common transaction of an "agent" with the centralized location in a topology of servers is verifying or substituting an address, the agent can also be used by a user for changing the users profile at the centralized location in a topology of servers. For example, a user changing his place of employment may specify a new daytime address. Likewise, a user switching to a new mail server system may designate this new email address as his address for evenings, weekends, or holidays.

Furthermore, according to the preferred embodiment of the present invention, **registering** of a current target address includes the agent **effecting** an interactive querying of a user at the user computer. For example, other data may be collected from a user, when the agent is downloaded, such as the users actual name, his physical domicile address, details of his personal profile or preferences, etc.

Furthermore, according to the preferred embodiment of the present invention, **registering** of a current target address includes **displaying** at the user computer information retrieved from the virtual centralized location in a topology of servers. For example, there may be ambiguities in the target address that is to registered. These ambiguities may be with regard to other address that have already been registered at the virtual centralized location in a topology of servers.

According to an embodiment of the present invention, (step b) **acquiring** a client software agent is by direct installation on the user computer; for example, from a diskette. According to an embodiment of the present invention, (step b) **acquiring** a client software agent is by remote download over a data-communication media to the user computer; for example, over the Internet. According to an embodiment of the present invention, (step b) **acquiring** a client software agent includes **authorizing** of an **installing** of a client software agent on another user computer, and the installing is by transfer of a copy of the client software agent; for example, by a site administrator to a plurality of interconnected computers under his administration.

According to an embodiment of the present invention, (step c) a client software agent **executing** an address substitution transaction includes:

- a **taking** of control over the processing of the electronic data-communications sending,
- an **intercepting** of at least one send-to address of the electronic data-communications sending,
- a **transmitting** to the virtual centralized location in a topology of servers of a verification query of the at least one intercepted address,
- a **receiving** from the virtual centralized location in a topology of servers of a response to the verification query,
- a **substituting** of each intercepted address for which the verification query response provides a target address that is different than the intercepted address, and
- a **returning** of control over the processing of the electronic data-communications sending.

Furthermore, according to the preferred embodiment of the present invention, **returning** of control over the processing includes a **transferring** of a message or of an electronic data-communications to an SMTP server of the user computer. This returning of control helps to achieve an additional measure of transparency.

According to an embodiment of the present invention, (step d) **executing** a response transaction includes **transmitting** a predetermined response to the user computer who originated the present verification query; for example a message saying that the user is on vacation and will only be "opening" his mail on his return. Furthermore, according to the preferred embodiment of the present invention, the response is a text message, a voice message, an audio content message, a visual content message, or any combination thereof.

According to an embodiment of the present invention, (step d) **executing** a response transaction includes searching a cache memory of the virtual centralized location in a topology of servers for a recently transmitted like-query response. This step can improve response time and thus improve apparent transparency. Often there are a series emails exchanged between two parties within a period of one or two hours. If a target address has been verified or substituted within the last hour or so and if the result of that transaction is still in local memory (e.g. cache), then little gain can be had from repeating the transaction. It is most probable that the result of the current transaction will be identical to that of the most recent like transaction.

According to an embodiment of the present invention, (step d-ii) **finding** a current target address in the list of registered addresses includes finding at least one mailing list having therein at least one of the extracted addresses. For example, a user knows a number of people who would be interested in a certain news item. He address the news item in a single email to all of these recipients via the agent of the present invention. The agent of the present invention transacts a finding of the mailing list having the largest number of these recipients as members. The user may the designate the entire mailing list in place of the few members of it whom he knows. This substitution probably gets the news item to many more interested people than putting a "pass this message along to friends" note at the header of the news item email.

According to an embodiment of the present invention, IF (step d-ii) the **finding** a current address in the list of registered address yields ambiguous results THEN the acquired client software agent **executing** an address substitution transaction including **notifying** of the user computer sending the electronic data-communications of the ambiguous results AND **accepting** a substitution preference from the user computer. This is another facile feature contributing to the agent's transparency. For example, a user wants to send an email to Joe. The agent transacts a verification/substitution communication with the virtual centralized location in a topology of servers. It is found that there are two Joes, Joe A and Joe B. The user chooses Joe B and further designates that until he designates otherwise, whenever he address Joe, it is his intention to direct that email to Joe B.

According to an embodiment of the present invention, (step d-iii) a virtual centralized location in a topology of servers **forming** a query response includes, in the query response, for substantially each intercepted address of the verification query that is not the current target address in the list of registered addresses, **listing** the current target address or addresses substantially as an acknowledgement. Simply stated, bouncing back the same address is saying "use it because we don't have a better address for you to try".

There are other variations of this problem of knowledge of a best possible target address. This problem is also compounded with aspects of operational transparency (minimal response time). Thus, also according to the preferred embodiment of the present invention, a virtual centralized location in a topology of servers **forming** a query response includes, in the query response:

- for substantially each intercepted address of the verification query that is likewise the current target address in the list of registered addresses, **listing** a confirmation acknowledgement, or
- for substantially each intercepted address of the verification query that is not listed in the list of registered addresses, **listing** a disclaimer acknowledgement or a message or data, or
- for substantially each intercepted address of the verification query that is listed in the list of registered addresses with an exception processing code, **listing** an acknowledgement according to the exception processing code.

The present invention also relates to an electronic data-communications response transaction system including a virtual centralized location in a topology of servers and interconnected therewith at least one user computer, wherein for each user computer originating a verification query the virtual centralized location in a topology of servers executes of a response transaction by:

- **extracting** each address of at least one intercepted addresses in the verification query,
- for substantially each extracted address, **finding** a current target address in a list of registered addresses,
- **forming** a query response of address acknowledgements substantially from the found current target addresses, and
- **transmitting** the formed query response to the user computer who originated the present verification query.

BRIEF DESCRIPTION OF THE DRAWINGS:

In order to understand the invention and to see how it may be carried out in practice, a preferred embodiment will now be described, by way of non-limiting example only, with reference to the accompanying drawings, in which:

Figures 1-3: illustrate schematic overviews of variations for implementing the method for resolving electronic addresses in data-communications addresses;

Figures 4-5: illustrate schematic views of the electronic data-communications response transaction system; and

Figure 6: illustrates a procedural view of intercepting an outgoing electronic data-communications message.

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT

Figures 1-3 illustrate schematic overviews of variations for implementing the method for resolving electronic addresses in data-communications addresses.

In figure 1, the user computer's mail software **11** (e.g. Netscape, Eudora, etc) views the client software agent **12** as if it is the closest SMTP server. For packets being sent from **11** to **12** having an e-mail address, a verification query & response is exchanged between **12** and a service center **13** wherein the e-mail address is verified as valid, substituted for according to a list of registered addresses at the service center, etc. and thereafter **12** incorporates the substance of the response from **13** into the original packet having an e-mail address, sending the possibly modified packet onto the actual substantially closest SMTP server **14**. The service center **13**, the closest SMTP server **14**, and packets further transferred from the SMTP server all reside in a data-communications topology **10** (e.g. the Internet). Packets not having an e-mail address are passed by **12** from **11** to **14** directly.

In figure 2, the user computer's mail software **21** (e.g. Microsoft outlook) views the client software agent **22** as a plug in. For packets being sent from **21** using **22** that have an e-mail address, a verification query & response is exchanged between **22** and a service center **23** wherein the e-mail address is verified as valid, substituted for according to a list of registered addresses at the service center, etc. and thereafter **22** incorporates the substance of the response from **23** into the original packet having an e-mail address, sending the possibly modified packet onto the actual substantially closest SMTP server **24**. The service center **23**, the closest SMTP server **14**, and packets further transferred from the SMTP server all reside in a data-communications topology **20** (e.g. the Internet).). Packets not having an e-mail address are passed by **22** from **21** to **24** directly.

In figure 3, the user computer's mail software **31** (e.g. Netscape, Eudora, etc) views the client software agent **32**, which is a front end to the substantially closest SMTP server **34**. For packets being sent from **31** to **32** having an e-mail address, a verification query & response is exchanged between **32** and a service center **33** wherein the e-mail address is verified as valid, substituted for according to a list of registered addresses at the service center, etc. and thereafter **32** incorporates the substance of the response from **33** into the original packet having an e-mail address, sending the possibly modified packet onto the actual substantially closest SMTP server **34**. The service center **33**, the client software agent **32**, the closest SMTP server **34**, and packets further transferred from the SMTP server all reside in a data-communications topology **30** (e.g. the Internet). Packets not having an e-mail address are passed by **32** from **31** to **34** directly.

Figures 4-5: illustrate schematic views of the electronic data-communications response transaction system. Figure 6 illustrates a procedural view of intercepting an outgoing electronic data-communications message.

A response transaction may be described as user computer mail software **61** sending packets found by **621** not containing an e-mail address are sent by the agent **620** to the substantially closest SMTP server **64**, while packets having an email address are held at **622** while the e-mail address transaction occurs.

At the service center **630** extracting each address of at least one intercepted addresses in the verification query occurs at **631**. Then for substantially each extracted address, **632** effects a finding for a current target address in a list of registered addresses, and **633** effects a forming of a query response of address acknowledgements substantially from the found current target addresses. The response being transmitted as a formed query response to **620** who substantially originated the present verification query.

Transmission is through the data-communications topology **60** to **622** where necessary e-mail addressing substitution into the original packet occurs. The packet is then sent along to **64**.

In figure 4, an electronic data-communications response transaction system including a service center **43** virtual centralized location in a topology of servers **40** and interconnected therewith at least one user computer **42**, wherein for each user computer originating a verification query the virtual centralized location in a topology of servers executes of the response transaction.

In figure 5, an electronic data-communications response transaction system including a virtual centralized location **531-535** in a topology of servers **50** and interconnected therewith at least one user computer **52**, wherein for each user computer originating a verification query the virtual centralized location in a topology of servers executes the heretofore described response transaction.

In the context of the preferred embodiment of the present invention, an "ActiveName" is an electronic data-communications address that is uniquely recognizable by the client software agent, and an "Inactive" e-mail address is any electronic data-communications address that does not comply with the condition of being the "uniquely recognizable by the client software agent". For example, an ActiveName is an e-mail address that starts with a plus sign (e.g. +someone@somewhere.something or +someone) or with an asterisks (e.g. *webmaster@company.org or *joe) or the like. Furthermore, and ActiveName member is an ActiveName having a registered an Inactive e-mail address, at a service center of a virtual centralized location I n a topology of servers.

The present invention relates to embodiments accomplishing and actualizing a method for directing electronic data-communications to a current target address, the method including the steps of:

a) at a virtual centralized location in a topology of servers, **establishing** a list of registered addresses;

b) each computer or a proxy processor thereto, of at least one user computers, **acquiring** a client software agent;

c) for substantially each occurrence of a user computer of the at least one user computers **61 sending** an electronic data-communications, the acquired client software agent **620 64** executing an address substitution **transaction 621** including a verification query **621 622** to the list of registered addresses; and

d) for substantially each occurrence of the virtual centralized location in a topology of servers receiving a verification query from a user computer of the at least one user computers, the virtual centralized location **630** in a topology of servers **60 executing** a response transaction, by:

i) **extracting 631** each address of the at least one intercepted addresses in the verification query,

ii) for substantially each extracted address, **finding 632** a current target address in the list of registered addresses,

iii) **forming 633** a query response of address acknowledgements substantially from the found current target addresses, and

iv) **transmitting 633** to **622** the formed query response to the user computer who originated the present verification query.

The following is one scenario of steps (c&d) logic used in the method of the present invention.

Mail Software at A User Computer

- A user creates a new mail message for sending and fills out all necessary data such as subject, body and recipients. In the recipient field the user can either specify an ActiveName or an Inactive e-mail address of one of the ActiveNames members.

- The user presses (double clicks) the send button.

- The mail software contacts an ActiveNames local smtp server (service center) that resides on the user's computer and starts negotiating for sending the mail.

Client Software Agent

- As soon as a connection is established from the mail software, the agent behaves as a regular smtp server to the mail software, and opens connection immediately to the original smtp server of the user.

- As soon as a relay connection is established between the mail software -> local smtp server -> smtp mail server, The agent relays all data between the client and the server while inspecting the content for addressing items.

- When an address item such as an ActiveName or an e-mail address is found, the relaying is suspended and a checking of the address with the service center occurs.

Service Center

- As soon as a request arrives in the form of "What is the Active e-mail address of (XXX@YY.COM or +MyActiveName)" the service center server queries a database for results.

- When results arrive it wraps up the result with predetermined protocol items and returns it to the client (user computer / agent).

Agent

- Now the agent receives the reply from the service center server that can contain either a new e-mail address or not.

- according to the preferred variation embodiment, if a new e-mail address is returned instead of an old e-mail address that was specified the agent prompts the user with a question of "the address you have specified points to a person that changed his e-mail address, to which address do you want to send: the old one or the new one?"

- If the user has selected to use the new address the agent intervenes in the stream of the mail software and replace the old address with an ActiveName.

- The agent then resumes the session.

Mail software

- The user returns to the mail screen of his mail software.

More Specifically – the software logic

The following are two scenarios of the method and system of the present invention's implementation logic, one with Microsoft outlook XX plug-in and the other is with smtp compliant mail software.

Microsoft Outlook - Mail Software

- The user creates a new mail message for sending and fills out all necessary data such as subject, body and recipients. In the recipient field the user can either specify an ActiveName or an Inactive e-mail address of one of the ActiveNames members.

- The user presses (or double clicks) the send button.

- Now the plug in (as in Fig 2 22) goes into Action. The plug in is a regular exchange compliant Microsoft extension and it has to be registered on the CheckNames event to get the control when the send occurs.

Plug in annotated source code:

```
/*
 * Copyright 1998 by OTUS, Inc.,
 * P.O. BOX 16072 Tel-Aviv, ISRSAEL
 * All rights reserved.
 *
 * This software is the confidential and proprietary information
 * of OTUS, Inc.
 * You shall not disclose such Confidential Information and shall use
 * it only in accordance with the terms of the license agreement you
 * entered into with OTUS.
 */

/* This trring should be in the registry for loading it */
/*
HKEY_LOCAL_MACHINE\Software\Microsoft\Exchange\Client\Extensio
ns

  ActiveNames =4.0;d:\program files\Microsoft visual
studio\myprojects\development\runtime\debug\bin\anexchext.dll;1;0000011
1111100

*/

// Pre definitions of What exchange extension we are implementing
#define INITGUID
#define USES_IID_IExchExt
```

```
#define USES_IID_IExchExtAdvancedCriteria
#define USES_IID_IExchExtAttachedFileEvents
#define USES_IID_IExchExtCommands
#define USES_IID_IExchExtMessageEvents
#define USES_IID_IExchExtPropertySheets
#define USES_IID_IExchExtSessionEvents
#define USES_IID_IExchExtUserEvents
#define USES_IID_IMessage
#define USES_PS_MAPI
#define USES_PS_PUBLIC_STRINGS

#include <afx.h>
#include <winsock.h>
#include <iostream.h>

#include "resrc1.h"
#include "EVNTEXT.H"
#include "resolver.h"
#include "traces.h"

//#include <INITGUID.H>
#include <MAPIGUID.H>


#define GET_CONFIG_OPCODE "GET_CONFIG"
extern unsigned short iWhatToDoFlag;
extern                    string serverIp;

#include "resolver.h"
```

```
static void     readConfiguration(void);
// this is the key to be registered
// 4.0;d:\program files\microsoft visual
studio\myprojects\development\runtime\debug\bin\anexchext.dll;1;0000011
1111100

////////////////////////////////////////////////////////////////////////////////
//  global data for DLL
HINSTANCE ghInstDLL = NULL;  // instance handle of DLL

////////////////////////////////////////////////////////////////////////////////
//  FUNCTION: DLLMain()
//
//  Purpose
//  Do initialization processing
//
//  Return Value
//  TRUE - DLL successfully loads and LoadLibrary will succeed.
//  FALSE - will cause an Exchange error message saying it cannot locate
//          the extension DLL.
//
//  Comments
//  We only need to get a copy of the DLL's HINSTANCE.
//
BOOL WINAPI DllMain(
   HINSTANCE  hinstDLL,
   DWORD  fdwReason,
   LPVOID  lpvReserved)
```

```
{
 if (DLL_PROCESS_ATTACH == fdwReason)
 {
   ghInstDLL = hinstDLL;
 }
 if (DLL_PROCESS_DETACH == fdwReason)
 {
   ghInstDLL = hinstDLL;
 }

 return TRUE;
}



////////////////////////////////////////////////////////////////////////////
//   FUNCTION: ExchEntryPoint
//
//   Parameters - none
//
//   Purpose
//   The entry point called by Exchange.
//
//   Return Value
//   Pointer to Exchange Extension (IExchExt) interface
//
//   Comments
//   Exchange Client calls this for each context entry.
//
```

```
LPEXCHEXT CALLBACK ExchEntryPoint(void)
{
        return new AnExchExt;
}


//////////////////////////////////////////////////////////////////////////////
//  AnExchExt::IdExchExt()
//
//  Parameters - none
//
//  Purpose
//  Comstructor.  Called during instantiation of AnExchExt object.
//
//
AnExchExt::AnExchExt()
{
 m_cRef = 1;

 m_pExchExtPropertySheets = new AnExchExtPropertySheets(this);
 m_pExchExtMessageEvents = new AnExchExtMessageEvents(this);

};



//////////////////////////////////////////////////////////////////////////////
// IExchExt virtual member functions implementation
```

```
//

////////////////////////////////////////////////////////////////////////
//   AnExchExtPropertySheets::Release()
//
//   Parameters - none
//
//   Purpose
//   Frees memory when interface is not referenced any more
//
//   Return value
//   reference count of interface
//

STDMETHODIMP_(ULONG) AnExchExt::Release()
{
 ULONG ulCount = --m_cRef;

 if (!ulCount)
 {
  delete this;
 }

return ulCount;

}

////////////////////////////////////////////////////////////////////////
//   AnExchExt::QueryInterface()
```

```
//
// Parameters
// riid   -- Interface ID.
// ppvObj -- address of interface object pointer.
//
// Purpose
// Returns a pointer to an interface object that is requested by ID.
//
// Comments
// The interfaces are requested everytime a new context is entered.  The
// IID_IExchExt* interfaces are ignored if not specified in the Exchange
// extensions registry.
//
// If an interface pointer is returned for more than one context, that
// interface is used by the client for each of those contexts.  Check the
// current context to verify if it is appropriate to pass back an interface
// pointer.
//
// Return Value - none
//

STDMETHODIMP AnExchExt::QueryInterface(REFIID riid, LPVOID
FAR * ppvObj)
{
  HRESULT hResult = S_OK;

  *ppvObj = NULL;

  if (( IID_IUnknown == riid) || ( IID_IExchExt == riid) )
```

```
{
  *ppvObj = (LPUNKNOWN)this;
}
else if (IID_IExchExtPropertySheets == riid)
{
 // if we are in the read or send context, do not return
 // propertysheet interface
  if ( (m_context == EECONTEXT_SENDNOTEMESSAGE)  ||
     (m_context == EECONTEXT_SENDPOSTMESSAGE)  ||
     (m_context == EECONTEXT_SENDRESENDMESSAGE) ||
     (m_context == EECONTEXT_READNOTEMESSAGE)  ||
     (m_context == EECONTEXT_READPOSTMESSAGE)  ||
     (m_context == EECONTEXT_READREPORTMESSAGE) )
       return E_NOINTERFACE;

 // otherwise return the interface
  *ppvObj = (LPUNKNOWN) m_pExchExtPropertySheets;
}
else if (IID_IExchExtMessageEvents == riid)
{
  *ppvObj = (LPUNKNOWN) m_pExchExtMessageEvents;
}
else
  hResult = E_NOINTERFACE;

if (NULL != *ppvObj)
  ((LPUNKNOWN)*ppvObj)->AddRef();

return hResult;
```

```
}



//////////////////////////////////////////////////////////////////////////
//   AnExchExt::Install()
//
//   Parameters
//   peecb     -- pointer to Exchange Extension callback function
//   eecontext -- context code at time of being called.
//
//   Purpose
//   Called once for each new context that is entered.
//
//   Return Value
//   S_OK - the installation succeeded for the context
//   S_FALSE - deny the installation fo the extension for the context
//
STDMETHODIMP AnExchExt::Install(LPEXCHEXTCALLBACK peecb,
ULONG eecontext, ULONG ulFlags)
{
  HRESULT hr;

  m_context = eecontext;


      // This is the IID for IOutlookGetObjectForExchExtCallback
      static const IID IID_IOnlyInOutlook =
{0x0006720D,0x0000,0x0000,{0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x4
6}};
```

```
        IUnknown* punk = NULL;
        hr = peecb->QueryInterface(IID_IOnlyInOutlook, (void**)&punk);
        if (punk)
                punk->Release();
        if (!SUCCEEDED(hr))
                return(S_FALSE);


    switch (eecontext)
    {
     case EECONTEXT_PROPERTYSHEETS:
     case EECONTEXT_SENDNOTEMESSAGE:
     case EECONTEXT_SENDPOSTMESSAGE:
     case EECONTEXT_SENDRESENDMESSAGE:
     case EECONTEXT_READNOTEMESSAGE:
     case EECONTEXT_READPOSTMESSAGE:
     case EECONTEXT_READREPORTMESSAGE:
         case EECONTEXT_ADDRBOOK:
       hr = S_OK;
       break;

     default:
       hr = S_FALSE;
       break;
    }


    return hr;
```

```
}

/////////////////////////////////////////////////////////////////////////////
// IExchExtPropertySheets virtual member functions implementation
//

/////////////////////////////////////////////////////////////////////////////
// AnExchExtPropertySheets::QueryInterface()
//
// Parameters
// riid  -- Interface ID.
// ppvObj -- address of interface object pointer.
//
// Purpose
// Return interface object upon request
//
// Return Value - none
//
// Comments
// Currently the Exchange client does not call QueryInterface from any
object
// except for IExchExt. This is implemented in case features are added to
// Exchange to require QueryInterface from any object. Also, as a "rule of
// OLE COM" this is the proper implementation of QueryInterface.
//

STDMETHODIMP AnExchExtPropertySheets::QueryInterface(REFIID
riid, LPVOID FAR * ppvObj)
{
    *ppvObj = NULL;
    if (riid == IID_IExchExtPropertySheets)
    {
      *ppvObj = (LPVOID)this;
      // Increase usage count of this object
      AddRef();
      return S_OK;
    }
    if (riid == IID_IUnknown)
    {
      *ppvObj = (LPVOID)m_pExchExt;  // return parent interface
      m_pExchExt->AddRef();
      return S_OK;
    }

    return E_NOINTERFACE;

}


///////////////////////////////////////////////////////////////////////////
//   AnExchExtPropertySheets::GetMaxPageCount()
//
//   Parameters
//   ulFlags -- a bitmask indicating what type of property sheet is being
//            displayed
//
//   Purpose
//  Returns the number of property pages which are to be added.
//
//  Return Value - maximum number of custom pages for the property
sheet
//
//  Exchange calls this to know how many PROPSHEETPAGE buffers it
needs
//  to allocate.
//

STDMETHODIMP_ (ULONG)
AnExchExtPropertySheets::GetMaxPageCount(ULONG ulFlags)
{
 ULONG ulNumExtSheets;

   switch (ulFlags)
   {
   // ignore these objects.
    case EEPS_FOLDER:
    case EEPS_MESSAGE:
      ulNumExtSheets = 0;
      break;

    case EEPS_STORE:
    case EEPS_TOOLSOPTIONS:
      ulNumExtSheets = 1;  // adding one propery page
      break;

    default:
        ulNumExtSheets = 0;
        break;
      }

    return ulNumExtSheets;
  }


  ////////////////////////////////////////////////////////////////////////
  //  AnExchExtPropertySheets::GetPages()
  //
  //  Parameters
  //  peecb   -- pointer to Exchange callback interface
  //  ulFlags -- a bitmask indicating what type of property sheet is being
  //             displayed
  //  ppsp    -- output parm pointing to pointer to list of property sheets
  //  pcpsp   -- output parm pointing to buffer containg number of property
  //             sheets actually used.
  //
  //  Purpose
  //  Fills the PROPSHEETPAGE members for the custom property page.
  //
  //  Return Value
  //  S_FALSE - do not add a new page
  //  S_OK - use the ppsp information for new pages.
  //
  //  Comments
  //  Exchange calls this method to gather information for any custom
  //  property pages to be added to the sheet. Here we are only adding
```

```
//   one.  ppsp may be an array of PROPSHEETPAGE structures to allow
you
//   to all multiple property pages.
//

STDMETHODIMP
AnExchExtPropertySheets::GetPages(LPEXCHEXTCALLBACK peecb,
               ULONG ulFlags, LPPROPSHEETPAGE ppsp, ULONG FAR
* pcpsp)
{
      LPMDB pMDB = NULL;
      LPMESSAGE pItem = NULL;

  *pcpsp = 0;


  // fill out members for the property page
  ppsp[0].dwSize = sizeof (PROPSHEETPAGE);
  ppsp[0].dwFlags = PSP_DEFAULT | PSP_HASHELP |
PSP_USEICONID;
  ppsp[0].hInstance = ghInstDLL;
  ppsp[0].pszTemplate = MAKEINTRESOURCE(IDD_OTUS);
  ppsp[0].pszIcon = MAKEINTRESOURCE(IDI_ACTIVENAMES);
  ppsp[0].pszTitle = "ActiveNames(tm)";
  ppsp[0].pfnDlgProc = (DLGPROC)OptionsDlgProc;
  ppsp[0].lParam = 0;
  ppsp[0].pfnCallback = NULL;
  ppsp[0].pcRefParent = NULL;
```

```
    *pcpsp = 1;

    return S_OK;
  }


  ///////////////////////////////////////////////////////////////////////////
  //  AnExchExtPropertySheets::FreePages()
  //
  //  Parameters
  //  ppsp -- pointer to a pointer to the first of a list of property pages
  //  cpsp -- number of custom property pages in the list
  //  ulFlags -- type of property page
  //
  //  Purpose
  //  Free any memory associated to the property sheet.
  //
  //  Return Value - none
  //
  //  Comments
  //  No parameters are used in this example but the function is used as
  //  a signal that the property sheet is going away and so memory may
  //  be freed.
  //

  STDMETHODIMP_ (VOID)
  AnExchExtPropertySheets::FreePages(LPPROPSHEETPAGE ppsp,
  ULONG ulFlags,
                      ULONG cpsp)
```

```
{
  // not used in this sample
}


///////////////////////////////////////////////////////////////////////////
// IExchExtMessageEvents virtual member functions implementation
//


///////////////////////////////////////////////////////////////////////////
//  AnExchExtMessageEvents::QueryInterface()
//
//  Parameters
//  riid   -- Interface ID.
//  ppvObj -- address of interface object pointer.
//
//  Purpose
//  Return interface object upon request
//
//  Return Value - none
//
//  Comments
//  Currently the Exchange client does not call QueryInterface from any
object
//  except for IExchExt.  This is implemented in case features are added to
//  Exchange to require QueryInterface from any object.  Also, as a "rule of
//  OLE COM" this is the proper implementation of QueryInterface.
//
```

```
STDMETHODIMP AnExchExtMessageEvents::QueryInterface(REFIID
riid, LPVOID FAR * ppvObj)
{

  *ppvObj = NULL;
  if (riid == IID_IExchExtMessageEvents)
  {
    *ppvObj = (LPVOID)this;
    // Increase usage count of this object
    AddRef();
    return S_OK;
  }
  if (riid == IID_IUnknown)
  {
    *ppvObj = (LPVOID)m_pExchExt;  // return parent interface
    m_pExchExt->AddRef();
    return S_OK;
  }
  return E_NOINTERFACE;

}



///////////////////////////////////////////////////////////////////////////////
//   AnExchExtMessageEvents::OnRead()
//
//   Parameters
//   lpeecb -- pointer to IExchExtCallback interface
```

```
//
//  Purpose
//  To extend or inhibit Exchange when displaying the send or read note
form.
//
//  Return Value
//  S_OK signals Exchange to not continue calling extensions
//  S_FALSE signals Exchange to continue calling extensions
//  Other MAPI Code errors will abort the send or read note form.
//
//

STDMETHODIMP
AnExchExtMessageEvents::OnRead(LPEXCHEXTCALLBACK lpeecb)
{
        return S_FALSE;
}

////////////////////////////////////////////////////////////////////
//  AnExchExtMessageEvents::OnReadComplete()
//
//  Parameters
//  lpeecb -- pointer to IExchExtCallback interface
//
//  Purpose
//  To do processing after message has been read.
//
//  Return Value
//  S_OK signals Exchange to not continue calling extensions
//   S_FALSE signals Exchange to continue calling extensions
//   Some MAPI Code error indicates a problem and will not display the
send
//   or read note form.
//
//   Comments.
//   If an error code, such as MAPI_E_CALL_FAILED, is returned,
Exchange will
//   call OnReadComplete again with the ulFlags parameter set to
//   EEME_COMPLETE_FAILED.  Returning the error code again will
cause Exchange
//   to not display the UI.
//

STDMETHODIMP
AnExchExtMessageEvents::OnReadComplete(LPEXCHEXTCALLBACK
lpeecb, ULONG ulFlags)
{

        return S_FALSE;
}

///////////////////////////////////////////////////////////////////////////
//   AnExchExtMessageEvents::OnWrite()
//
//   Parameters
//   lpeecb -- pointer to IExchExtCallback interface
//
//   Purpose
// This method is called when a message is about to be written. The
message
// only has default properties at this point. It does not contain
// properties which the user has added by way of recipients, subject,
// message text, or attachments.
// This method is called when the user Sends or Saves a message
//
// Return Value
// S_OK signals Exchange to not continue calling extensions
// S_FALSE signals Exchange to continue calling extensions
//
//

STDMETHODIMP
AnExchExtMessageEvents::OnWrite(LPEXCHEXTCALLBACK lpeecb)
{
        return S_FALSE;
}

STDMETHODIMP
AnExchExtMessageEvents::OnWriteComplete(LPEXCHEXTCALLBACK
lpeecb, ULONG ulFlags)
{
        return S_FALSE;
}


///////////////////////////////////////////////////////////////////////////
// AnExchExtMessageEvents::OnSubmit()
```

```
//
// Parameters
// lpeecb -- pointer to IExchExtCallback interface
//
// Purpose
// Called before message data has been written and is
// is submitted to MAPI.
//
// Return Value
// S_OK signals Exchange to not continue calling extensions
// S_FALSE signals Exchange to continue calling extensions
//
// Set a member function to show that submit has been called
// to indicate to OnWriteComplete that the user has hit the
// Send button and is not just saving the message.
//

STDMETHODIMP
AnExchExtMessageEvents::OnSubmit(LPEXCHEXTCALLBACK lpeecb)
{       //We are adding this handling under submit for other exchange 4.0
        return(S_FALSE);
}

///////////////////////////////////////////////////////////////////////////
// AnExchExtMessageEvents::OnSubmitComplete()
//
// Parameters
// lpeecb -- pointer to IExchExtCallback interface
//
```

```
// Purpose
// Called after message has been submitted to MAPI.
//
// Return Value - none
//
// A flag is cleared to indicate to other methods that Exchange is
// finished submitting the message.
//

STDMETHODIMP_ (VOID)
AnExchExtMessageEvents::OnSubmitComplete(LPEXCHEXTCALLBAC
K lpeecb, ULONG ulFlags)
{
}




STDMETHODIMP
AnExchExtMessageEvents::OnCheckNames(LPEXCHEXTCALLBACK
lpeecb)
{
        HRESULT hr = S_FALSE;
        LPADRLIST pAdrList = 0, pNewAdrList = 0;
        // Get the current recipient list
        hr = lpeecb->GetRecipients(&pAdrList);
        if (FAILED(hr) || (pAdrList == 0))        return(S_FALSE);
```

```
try
{
        // Initiate the result with a failed constant
        unsigned short resolveResult = RESOLVED_FAILED;
        resolveResult = OtResolveAddressList(lpeecb, pAdrList,
&pNewAdrList);
        if (resolveResult == RESOLVED_FAILED)      // Did we
failed resolving
        {
                FreePadrlist(pAdrList);
                return(S_FALSE);
        }

        // Try to do another resolve by microsoft
        // Try again to resolve all the new entries
        // to resolve all the changes we have done in the list
        hr = ResolveByExchange(pNewAdrList, false);
        if (hr != S_OK)
                resolveResult = RESOLVED_PARTIALY;        // The
list has been resolved partially

        // Update outlook with the windows settings
        hr = lpeecb->SetRecipients(pNewAdrList);
        // Set focus to the window after java might have taken it
        HWND        currentMailWnd = 0;
        if (lpeecb->GetWindow(&currentMailWnd ) == S_OK)
        {
                if (currentMailWnd != 0)
```

```
                        SetForegroundWindow(currentMailWnd);
                }
                // Cleanup
                OtClearPointers();
                FreePadrlist(pAdrList);

                // Anayzlye result
                if (FAILED(hr))
                        return(S_FALSE);
                else
                {
                        // Did we resolved only partially ?
                        // Outlook 98 dumps when you return ok when there
are bad recipients
                        if (resolveResult == RESOLVED_PARTIALY)
                                return(S_FALSE);
                        else
                                return(S_OK);
                }
        }
        catch (...)
        {
                OtClearPointers();
                if (pAdrList != 0)
                        FreePadrlist(pAdrList);
        }
        return(S_FALSE);
}
```

```
//////////////////////////////////////////////////////////////////////
//  AnExchExtMessageEvents::OnCheckNamesComplete()
//
//  Parameters
//  lpeecb -- pointer to IExchExtCallback interface
//
//  Purpose
//  Called after exchange has completed resolving names in the message
//  recipients table.
//
//  Return Value
//  S_OK signals Exchange to not continue calling extensions
//  S_FALSE signals Exchange to continue calling extensions
//

STDMETHODIMP
AnExchExtMessageEvents::OnCheckNamesComplete(LPEXCHEXTCAL
LBACK lpeecb, ULONG ulFlags)
{
        return(S_OK);
}
```

- In the code above a service center (e.g. **630** of figure 6) intercepts the call and call an internal library to find out what is the correct addresses, if the agent (e.g. 620) detects a change then it asks from outlook to accept the changes.

Mail software

- The user returns to the mail screen of his mail software.

smtp compliant mail software: below is a sample of how the system of the present invention is implemented by a Local smtp server (written in java).

Mail Software

- The user creates a new mail message for sending and fills out all necessary data such as subject, body and recipients. In the recipient field he can either specify an ActiveName or an Inactive e-mail address of one of the ActiveNames memebers.

- The user presses the send button.

- The mail software contacts the service center local smtp server that resides on the user's computer and starts negotiating for sending the mail.

Agent

- As soon as a connection is established from the mail software, the agent behaves as a regular smtp server to the mail software, and opens connection immediately to the original smtp server of the user.

- As soon as a relay connection is established between the mail software -> local smtp server -> smtp mail server, The agent relays all data between the client and the server while inspecting the content for addressing items.

- When the agent finde an address item such as an ActiveName or an e-mail address the it suspends the relaying and begins a process for checking the address with the service center for any changes.

Service Center server

- As soon as a request arrives in the form of "What is the Active e-mail address of (XXX@YY.COM or +MyActiveName)" the service center server queries the database for results.

- When results arrive it wrapps up the result with our protocol items and returns it to the client.

Agent

- Now the agent receives the reply from the service center that can contain either a new e-mail address or not.

- Preferably, if the reply has a new e-mail address instead of an old e-mail address that was specified the agent prompts the user with a question of "the address you have specified points to a person that changed his e-mail address, to which address do you want to send: the old one or the new one?"

- If the user selects to use the new address the agent intervenes in the stream of the mail software and replace the old address with the selected one.

- the agent then resumes the session.

Mail software

- The user returns to the mail screen of his mail software.

```
public void run()
{
        // First check several parameters
        if (iSocket == null)  return;

        loadConfiguration();

        if ((iTargetSmtp == null) ||
                (iTargetSmtp.length() == 0))
            {
                try
                {
                    if (iSocket != null)
                    {
                        iSocket.close();
                        iSocket = null;
                    }

                }
                catch(IOException en)
                {
                    if (OtGeneral.logging)
                        OtGeneral.systemLog("Error closing the
smtp sockets", en);
                }
                return;
            }

            SocketrelayingSocket = null;
            // Do all the relaying loop between the client and the server
            try
            {
                BufferedReader inSmtpClient  = new
BufferedReader(new InputStreamReader(iSocket.getInputStream(),
"8859_1"));
```

```
                    BufferedWriter outSmtpClient = new
BufferedWriter(new OutputStreamWriter(iSocket.getOutputStream(),
"8859_1"));

                    // try to connect to the client's smtp server
                    // Check if we are using socks server
                    if (OtServerProxy.usingSocks())
                    {
                         try       // Check if we can connect to the address
by socks
                         {
                              relayingSocket =
OtServerProxy.getSocket(iTargetSmtp, iSmtpPort);
                         }
                         catch(Exception ex)
                         {
                              relayingSocket = null;
                         }
                         // Check if we failed by socks and we retrying
without socks
                         if (relayingSocket == null)
                              relayingSocket  = new
Socket(iTargetSmtp, iSmtpPort);
                    }
                    else
                         relayingSocket  = new Socket(iTargetSmtp,
iSmtpPort);
```

```
                    BufferedReader inSmtpServer  = new
BufferedReader(new InputStreamReader(relayingSocket.getInputStream(),
"8859_1"));
                    BufferedWriter outSmtpServer = new
BufferedWriter(new
OutputStreamWriter(relayingSocket.getOutputStream(), "8859_1"));

                    iLastResolvedAddresses = new Hashtable();
                    String  recievedBuff= null;
                    // This main loop is the SMTP state machine
                    do
                    {
                         // Get SMTP Server data and forward it to the
client
                         do
                         {
                           recievedBuff = hear(inSmtpServer, "From
smtp server");
                              say(outSmtpClient, recievedBuff, "To
smtp client");
                         } while (inSmtpServer.ready());

                         // Check if the server has responded to a quit
message
                         if ((recievedBuff != null) &&
                              (recievedBuff.length() > 3) &&
                              (recievedBuff.startsWith("221")))
                              throw new IOException("Connection
was closed after quit");
```

```
                        // Get the SMTP Client data and forward it to
the server
                        do
                        {
                recievedBuff = hear(inSmtpClient, "From smtp
client");
                                // Check if the user wants a signature at
the end of
                                // the data state
                                if (iDataState && iSignature)
                                {
                                        // Check if it is the end of the data
session
                                        if
(recievedBuff.equalsIgnoreCase("."))
                                                say(outSmtpServer,
"ActiveName: " + OtActiveNames.ACTIVE_NAMES_PREFIX +
iActiveName,
                                                        "Sending
signature");

                                }

                                // Check if we have any address for
resolving
                                if (!iDataState)
```

```
                                        recievedBuff =
processClientInput(recievedBuff);

                                else
                                        recievedBuff =
processMimeTypes(recievedBuff);



                                // Forward the data to the user
                                say(outSmtpServer, recievedBuff, "To
smtp server");
                                // Check if we have entered or exited
from data state
                                if (iDataState)
                                        checkDataState(recievedBuff);
                        } while (inSmtpClient.ready() || iDataState);

                        // Check if have exited from the data state
                        checkDataState(recievedBuff);

                } while (true);
        }
        catch(Exception e)
        {
                try
                {
                        if (iSocket != null)
                        {
                                iSocket.close();
                        iSocket = null;
                    }
                }
                catch(IOException en)
                {
                    if (OtGeneral.logging)
                        OtGeneral.systemLog("Error closing the
smtp sockets", en);
                }
                try
                {
                    if (relayingSocket != null)
                    {
                        relayingSocket.close();
                        relayingSocket = null;
                    }
                }
                catch(IOException en)
                {
                    if (OtGeneral.logging)
                        OtGeneral.systemLog("Error closing the
smtp sockets", en);
                }
            }
            iLastResolvedAddresses = null;
        }
```

- This is the function that parses the data

```
        private StringprocessClientInput(String recievedBuff)
        {
                String tempBuff = recievedBuff.toUpperCase();
                tempBuff = tempBuff.trim();

                // Check if we encountered a recipt to line
                if (tempBuff.startsWith("RCPT TO:") == false)
    return(recievedBuff);

                // Build the email address to resolve
                String emailAddress = null;
                String startOfLine  = null;
                String endOfLine    = null;
                 // Check how to parse the recipient name <XXX> or XXXX
                if (tempBuff.indexOf("<") != -1)
                {

                        startOfLine  = recievedBuff.substring(0,
    recievedBuff.indexOf("<") + 1);
                        emailAddress =
    recievedBuff.substring(recievedBuff.indexOf("<") + 1,

            recievedBuff.lastIndexOf(">"));
                        endOfLine    = ">";
                }
                else
                {
                        startOfLine = "RCPT TO:";
                emailAddress = recievedBuff.substring("RCPT
TO:".length());
                endOfLine = "";
            }

            // Replace the old line with the new recipient
            if (emailAddress != null)
            {
                String newAddress = resolveAddress(emailAddress);
                if (OtGeneral.logging)
                    OtGeneral.debugLog("Address:" +
emailAddress + " has changed to " + newAddress);
                recievedBuff = startOfLine + newAddress +
endOfLine;
            }
            return(recievedBuff);
        }
```

Numbers, alphabetic characters and roman symbols are designated in the following claims for convenience of explanations only and should by no means be regarded as imposing particular order on the method steps.

The present invention has been described with a certain degree of particularity, however those versed in the art will readily appreciate that various modifications and alterations may be carried out without departing from the spirit and scope of the following claims:

CLAIMS:

1. A method for directing electronic data-communications to a current target address, the method comprising the steps of:

(a) at a virtual centralized location in a topology of servers, **establishing** a list of registered addresses;

(b)each computer or a proxy processor thereto, of at least one user computers, **acquiring** a client software agent;

(c) for substantially each occurrence of a user computer of the at least one user computers **sending** an electronic data-communications, the acquired client software agent **executing** an address substitution transaction including a verification query to the list of registered addresses; and

(d)for substantially each occurrence of the virtual centralized location in a topology of servers receiving a verification query from a user computer of the at least one user computers, the virtual centralized location in a topology of servers **executing** a response transaction, by:

(i) **extracting** each address of the at least one intercepted addresses in the verification query,

(ii) for substantially each extracted address, **finding** a current target address in the list of registered addresses,

(iii) **forming** a query response of address acknowledgements substantially from the found current target addresses, and

(iv) **transmitting** the formed query response to the user computer who originated the present verification query.

2. The method according to claim 1 wherein a client software agent **executing** an address substitution transaction includes:

(a) a **taking** of control over the processing of the electronic data-communications sending, (b)an **intercepting** of at least one send-to address of the electronic data-communications sending,

(c) a **transmitting** to the virtual centralized location in a topology of servers of a verification query of the at least one intercepted address,

(d)a **receiving** from the virtual centralized location in a topology of servers of a response to the verification query,

(e) a **substituting** of each intercepted address for which the verification query response provides a target address that is different than said intercepted address, and

(f) a **returning** of control over the processing of the electronic data-communications sending.

3. The method according to claim 1 wherein a virtual centralized location in a topology of servers **forming** a query response includes, in the query response, for substantially each intercepted address of the verification query that is not the current target address in the list of registered addresses, **listing** the current target address or addresses substantially as an acknowledgement.

4. The method according to claim 3 wherein a virtual centralized location in a topology of servers **forming** a query response includes, in the query response:

(a) for substantially each intercepted address of the verification query that is likewise the current target address in the list of registered addresses, **listing** a confirmation acknowledgement, or

(b)for substantially each intercepted address of the verification query that is not listed in the list of registered addresses, **listing** a disclaimer acknowledgement or a message or data, or

(c) for substantially each intercepted address of the verification query that is listed in the list of registered addresses with an exception processing code, **listing** an acknowledgement according to the exception processing code.

5. The method according to claim 1 wherein **acquiring** a client software agent includes **transacting** a **registering** of a current target address and of other predetermined data for a user at a user computer.

6. The method according to claim 5 wherein the client software agent **transacting** includes, for a user at a user computer, the virtual centralized location in a topology of servers **updating** of a registered current target address of other predetermined data in the established list of registered addresses.

7. The method according to claim 1 wherein **acquiring** a client software agent is by direct installation on the user computer.

8. The method according to claim 1 wherein **acquiring** a client software agent is by remote download over a data-communication media to the user computer.

9. The method according to claim 1 wherein **acquiring** a client software agent includes **authorizing** of an **installing** of a client software agent on another user computer, and said installing is by transfer of a copy of said client software agent.

10. The method according to claim 5 wherein **registering** of a current target address includes the agent **effecting** an interactive querying of a user at the user computer.

11. The method according to claim 5 wherein **registering** of a current target address includes **displaying** at the user computer information retrieved from the virtual centralized location in a topology of servers.

12. The method according to claim 2 wherein **returning** of control over the processing includes a **transferring** of a message or of an electronic data-communications to an SMTP server of the user computer.

13. The method according to claim 1 wherein a current target address in the list of registered addresses is a mailing list having at least one address.

14. The method according to claim 1 wherein **finding** a current target address in the list of registered addresses includes finding at least one mailing list having therein at least one of the extracted addresses.

15. The method according to claim 1 wherein **establishing** a list of registered addresses includes **managing** at least one address therein according to time-of-day, day-of-week, date-in-year, or a personal schedule.

16. The method according to claim 1 wherein **executing** a response transaction includes **transmitting** a predetermined response to the user computer who originated the present verification query.

17. The method according to claim 16 wherein the response is a text message, a voice message, an audio content message, a visual content message, or any combination thereof.

18. The method according to claim 1 wherein **executing** a response transaction includes searching a cache memory of the virtual centralized location in a topology of servers for a recently transmitted like-query response.

19. The method according to claim 1 wherein, for segments not containing an electronic data-communications address, the client software agent passes segments to a server using a simple mail protocol.

20. The method according to claim 1 wherein a user's view of acquired client software agent operation emulates SMTP server operation.

21. The method according to claim 1 wherein a user's view of acquired client software agent operation emulates a plug-in electronic data-communications software package's operation.

22. The method according to claim 1 wherein IF the **finding** a current address in the list of registered address yields ambiguous results THEN the acquired client software agent **executing** an address substitution transaction including **notifying** of the user computer sending the electronic data-communications of the ambiguous results AND **accepting** a substitution preference from the user computer.

23. An electronic data-communications response transaction system comprising a virtual centralized location in a topology of servers and interconnected therewith at least one user computer, wherein for each user computer originating a verification query said virtual centralized location in a topology of servers executes of a response transaction by:

(a) **extracting** each address of at least one intercepted addresses in the verification query,

(b)for substantially each extracted address, **finding** a current target address in a list of registered addresses,

(c) **forming** a query response of address acknowledgements substantially from the found current target addresses, and

(d)transmitting the formed query response to the user computer who originated the present verification query.

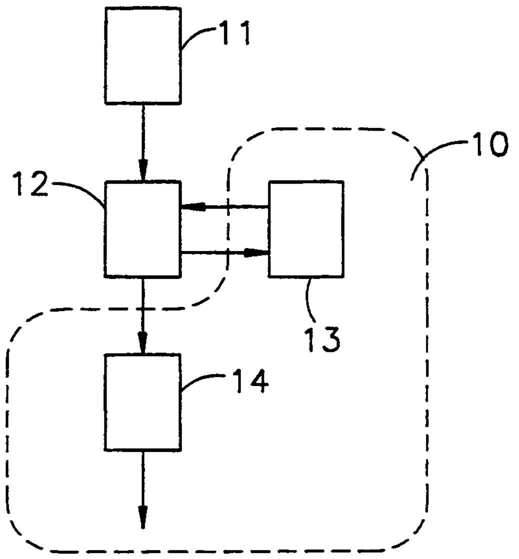


FIG.1

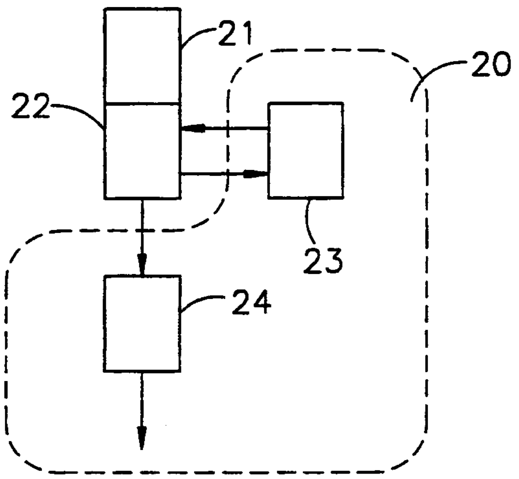


FIG.2

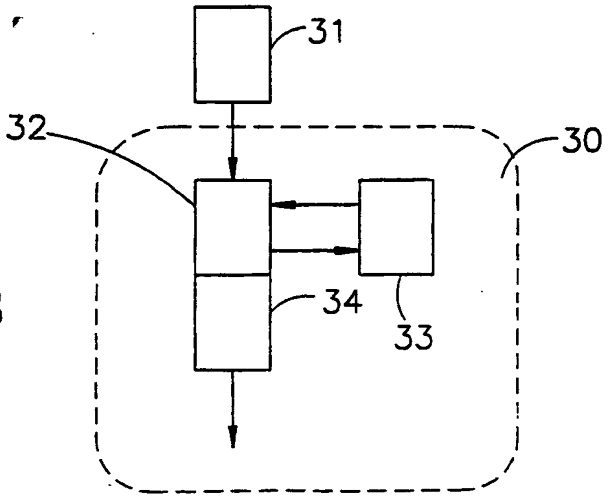


FIG.3

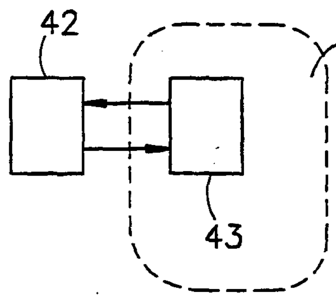


FIG.4

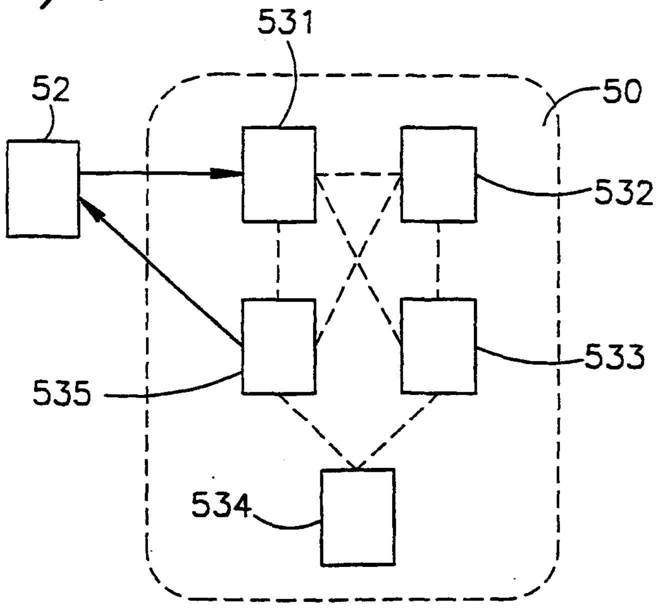


FIG.5

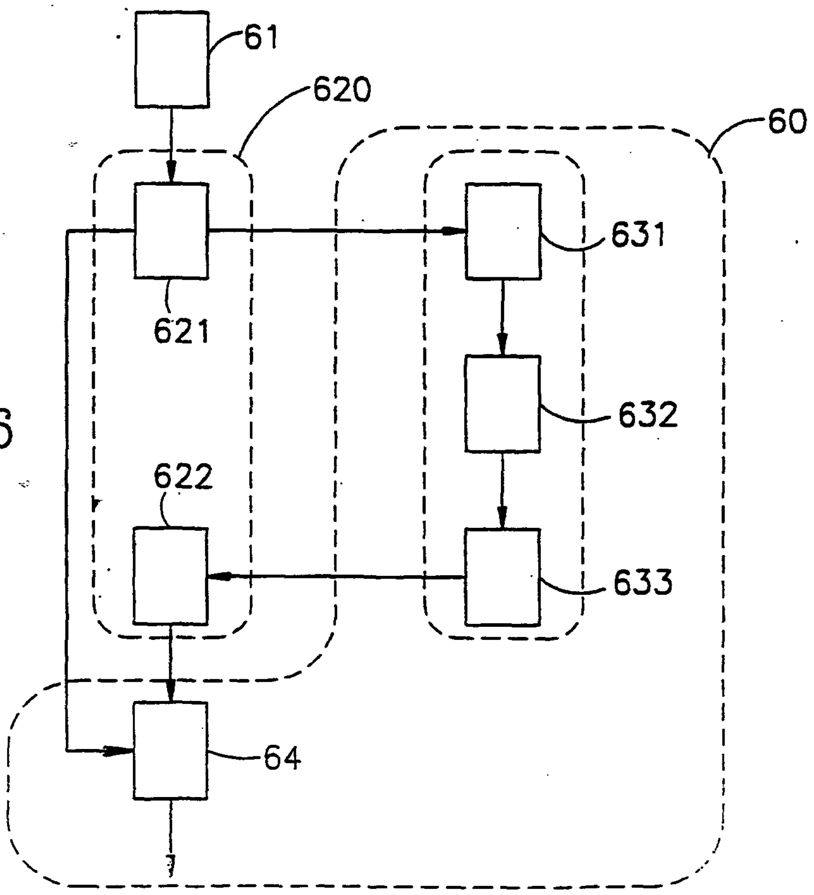


FIG.6

INTERNATIONAL SEARCH REPORT

International Application No
PCT/IL 99/00318

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 H04L12/58 G06F17/60

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 H04L G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 5 826 039 A (JONES MARK ALAN) 20 October 1998 (1998-10-20) figure 2 column 4, line 21 -column 9, line 67	1-23
X	WO 99 17241 A (ERICSSON GE MOBILE INC) 8 April 1999 (1999-04-08) page 7, line 1 -page 13, line 13 figures 1-9	1-23
X	PATENT ABSTRACTS OF JAPAN vol. 1999, no. 10, 31 August 1999 (1999-08-31) & JP 11 134267 A (NIPPON TELEGR &AMP;TELEPH CORP &LT;NTT&GT;), 21 May 1999 (1999-05-21) abstract	23
A		1-22

☐ Further documents are listed in the continuation of box C.

[✓ X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier document but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.
"&" document member of the same patent family

Date of the actual completion of the international search 18 May 2000	Date of mailing of the international search report 26/05/2000
Name and mailing address of the ISA European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Authorized officer Eraso Helguera, J

INTERNATIONAL SEARCH REPORT

Information on patent family members

Inter nal Application No
PCT/IL 99/00318

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5826039 A	20-10-1998	CA 2191505 A	30-06-1997
WO 9917241 A	08-04-1999	AU 9311198 A	23-04-1999
JP 11134267 A	21-05-1999	NONE	